НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2019р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 6.050103 “ Програмна інженерія ”

на тему «Засоби доступу до реєстру інформаційних ресурсів в хмарному сервісі Google Drive»

Виконав (-ла): студент (-ка) 4 курсу, групи ТВ-51

Гриявець Олександр Юрійович _____
(прізвище, ім’я, по батькові) (підпис)

Керівник старший викладач Колумбет В.П. _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2019 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки 6.050103 “Програмна інженерія”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2019р.

ЗАВДАННЯ
на дипломну роботу студенту

Гирявець Олександр Юрійович
(прізвище, ім’я, по батькові)

1. Тема роботи «Засоби доступу до реєстру інформаційних ресурсів в хмарному сервісі Google Drive»
керівник роботи Колумбет Вадим Петрович старший викладач
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”22”05 2019р. № 1325С

2. Строк подання студентом роботи 17.06.2019-21.06.2019
3. Вихідні дані до роботи платформа Visual Studio Code v1.27, мова програмування Javascript з фреймворком Vue.js для клієнтської та фреймворокм express для серверної
сторони

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Розробити алгоритм для моделювання гідроакустичного сигналу. Реалізувати код Web-сервісу для цього алгоритму. Створити зручний користувацький інтерфейс для перегляду, створення, редагування та надання рівнів доступу до інформаційних ресурсів

_____

5. Перелік ілюстративного матеріалу

«Постановка задачі», «Сфери застосування даного рішення», «Існуючі рішення», «Використані технології», «Архітектура системи», «Головний екран Web-сервісу», «Головні елементи інтерфейсу», «Введення нових направлень», «Алгоритм генерації сигналу», «Кінцевий результат», «Необхідно для користування», «Висновки».

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання ”14” жовтня 2019 р.

АНОТАЦІЯ

Метою роботи було створення веб-сервісу для надання доступу до реєстру інформаційних ресурсів в хмарному сервісі Google Drive. Для досягнення поставленої мети було розроблено серверний застосунок, який дозволяє завантажувати файли до хмарного сервісу Google Drive та користувацький інтерфейс, який дозволяє користувачеві переглядати список інформаційних ресурсів, створювати їх, редагувати та налаштовувати рівні доступу як до конкретного ресурсу, так і до групи ресурсів, які знаходяться в межах однієї директорії.

Записка містить 51 сторінок, 14 рисунків та 10 посилань.

ABSTRACT

The purpose of the work was to create a web service to provide access to the register of information resources in the cloud service Google Drive. To achieve the goal, has been developed a server application that allows you to upload files to the Google Drive cloud service and a user interface that allows the user to browse the list of information resources, create them, edit and configure access levels for both a particular resource and a group of resources that are within the same directory.

The note contains 51 pages, 14 images and 10 references.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

- CIP - Система інформаційних ресурсів
- JS - JavaScript
- SASS - Syntctically Awesome Style Sheets
- ОС - операційна система
- IDE - Integrated Development Environment
- API - Application Progamming Interface

ВСТУП

В умовах формування в країні основ інформаційного суспільства особливої важливості набуває розбудова Систем електронних інформаційних ресурсів (СІР), метою яких є створення інструменту, здатного підвищити ефективність використання інформаційних ресурсів та істотно знизити витрати на їх утримання за рахунок використання метаданих щодо інформаційних масивів, баз даних, реєстрів та інших видів інформаційних ресурсів (ІР), у першу чергу тих, що утворюються за рахунок держбюджету в органах влади та інших державних установах та організаціях.

Проблематика формування СІР пов’язана, перш за все, з тим, що наявні автоматизовані інформаційні системи, а також бази даних є організаційно розпорошеними та функціонально роз’єднаними і не забезпечують інформаційну взаємодію; створення та експлуатація баз даних, надання інформації, її використання та передача не забезпечені організаційно, не підкріплені належною базою; не існує єдиних правил та методів доступу до інформаційних ресурсів, які б забезпечували якісну інформаційну взаємодію, в тому числі з урахуванням вимог інформаційної безпеки.

Питанням розбудови системи інформаційних ресурсів останнім часом присвячується усе більше публікацій. Але слід зазначити, що хоча концептуально питання щодо інтеграційних процесів у цій сфері були поставлені, досліджені вони ще недостатньо. Також відсутня й методологія цих процесів.

1 ПОСТАНОВКА ЗАДАЧІ

Метою дипломної роботи є створення веб-сервісу для надання доступу до реєстру інформаційних ресурсів. Такий сервіс повинен виконувати такі завдання:

1. Забезпечувати реєстрацію та авторизацію користувачів.
2. Проводити надійну аутентифікацію користувачів.
3. Надавати змогу створювати інформаційні ресурси.
4. Надавати змогу переглядати інформаційні ресурси.
5. Структурувати інформаційні ресурси по деректоріям.
6. Встановлювати рівні доступу до інформаційних ресурсів.
7. Вивантажувати дані на локальний комп’ютер.
8. Зберігати метадані щодо конкретного інформаційного ресурсу.
9. Завантажувати файли до хмарного сховища Google Drive.

Отже, для виконання поставленої задачі необхідно зробити наступні кроки:

1. Дослідити поняття інформаційного ресурсу та принципи створення CIP.
2. Проаналізувати наявні системи, визначити їх переваги та недоліки.
3. Розробити методи взаємодії з API хмарного сховища Google Drive для додавання та видалення файлів.
4. Визначити схему метаданих інформаційного ресурсу.
5. Створити зручний інтерфейс для роботи з реєстром інформаційних ресурсів.

Структурно система повинна бути розподілена на окремі незалежні, легкозамнювані модулі, які взаємодіють між собою.

2 АНАЛІЗ РЕЄСТРУ ІНФОРМАЦІЙНИХ РЕСУРСІВ

Розглянемо поняття електронного інформаційного ресурсу та реєстру таких ресурсів. А також проаналізуємо наявні приклади реалізації таких систем.

2.1 Поняття електронного інформаційного ресурсу

У найзагальнішому випадку поняття електронного ресурсу можна визначити як будь-яку інформацію, для відтворення якої необхідні електронні пристрої. При такому визначенні поняття електронного ресурсу не містить вказівок ні на тип інформації яка відтворюється, ні на її утримання.

Детальніший розгляд поняття електронних ресурсів (ЕР) передбачає, що ЕР – це сукупність програмних засобів, інформаційних, технічних, нормативних та методичних матеріалів, повнотекстових електронних видань, включаючи відео та аудіоматеріали, ілюстративні матеріали та каталоги електронних бібліотек, розміщені на інформаційних носіях та в мережі Інтернет [10]. В загальному, до поняття ЕР можна віднести також навчальні відеофільми та звукозапису, для відтворення яких досить звичайного магнітофона або CD-плеєра, найбільшу увагу приділено саме ресурсам, призначеним для відтворення на комп'ютерах або сумісних з ними пристроях – електронних книжках. Пристрої, які називаються електронними книгами (рідерами) являють собою різновид комп'ютерів, призначених для відтворення текстової інформації, яка представлена в електронному вигляді і характеризуються обмеженим функціоналом, тому, якщо не сказано інше, будемо припускати, що ЕР признаіені для відтворення на повнофункціональних комп'ютерах.

Розглянемо також інші визначення поняття «інформаційні ресурси». Так, у монографії «Е-будущее и информационное право» приведено наступне визначення: «сукупність інформаційних продуктів певного призначення, необхідних для забезпечення інформаційних потреб споживачів у визначеній сфері діяльності». О. В. Соснін зазначає: «Найбільш повно характеризувати інформаційний ресурс (ІР) можна як інформацію, створену чи виявлену, зареєстровану і оцінену. ІР набуває особливих властивостей, сутність яких і робить його інформаційним продуктом для споживання».

Звідси видно, у цих визначеннях спільним є те, що вони наголошують на споживчій властивості інформаційного ресурсу, як інформаційного продукту. Однак, у О. В. Сосніна цей продукт є заснованим на інформації, яка раніше була виявлена та оцінена: «Для того, щоб стати інформаційним ресурсом, потоки інформації повинні мати деякі специфічні якості, завдяки яким вони стають соціально значущими, технологічно придатними, тобто такими, що мають цінність для практичного застосування. Такою основною властивістю є системна організованість (організація) інформаційних потоків та їх окремих елементів».

Аналізуючи вищезгадані визначення інформаційних ресурсів, можна виявити, що головною їх відмінністю від визначення, що декларовано Законом України «Про Національну програму інформатизації», є відмова від використання терміну «документ». Тобто, це продукт споживання, що ґрунтується на раніше зафіксованій інформації (за О. В. Сосніним), але без акценту на тому, що зазначена фіксація інформації є, по суті, її документуванням. Незважаючи на це, говориться про «системну організацію інформаційних потоків та їх окремих елементів». По суті, процес «створення чи виявлення, реєстрування» інформації та подальше її «споживання» можна описати як зберігання та передача інформації, зафіксованої на фізичному носії в часі та просторі, що є основною функцією документа за визначенням ДСТУ 2732:2004, а «системну організацію інформаційних потоків та їх окремих елементів» як «запис інформації», що має відповідати властивостям певного жанру чи номіналу.

Жанрові властивості запису інформації – це функційні та структурно-

композиційні ознаки певного жанру твору літератури чи мистецтва (роман, монографія, хронікально-документальний фільм тощо).

Номінальні властивості запису інформації – це функційні та структурно композиційні ознаки певного виду задокументованої службової чи приватної інформації (наказ, акт, протокол, лист, щоденник тощо)». Невизначеність з фіксацією інформації на носії інформації та чітких жанрових та номінальних характеристик для інформаційних ресурсів створює передумови для подальшої невизначеності в цілому інформаційних ресурсів у забезпечені будь-якої діяльності.

Одним із основних чинників, що спричиняє такий підхід, є впровадження у діяльність інформаційних технологій. Так О. В. Соснін зазначає, що «основними сучасними формами організації інформаційного ресурсу є: файл, база даних, банк даних, база знань, бібліотека та ін.» При цьому вчений не уточнює про які саме бази даних і знань та банки даних йде мова. У Законі України «Про Концепцію Національної програми інформатизації» йдеться про створення таких державних інформаційних ресурсів (на час його прийняття): «бази даних «Законодавчі та нормативні акти України», «Податки України» (занесено близько 10 тисяч документів), «Ресурси України» (постійно актуалізується інформація про 260 тисяч підприємств та організацій України), «Єдиний державний реєстр підприємств та організацій України» (занесено з присвоєнням унікального коду більше ніж 600 тисяч суб'єктів господарської діяльності)». За 14 років, після прийняття вищезгаданого закону, ці показники значно збільшилися. Про це говорять нормативно-правові акти України, які регулюють створення та використання електронних інформаційних ресурсів.

2.2 Поняття реєстру та приклади реалізації систем інформаційних ресурсів

Реєстр – це інформаційно-телекомунікаційна система, яка призначена для реєстрації, обліку, накопичення, оброблення і зберігання відомостей про склад, зміст, розміщення, умови доступу до електронних інформаційних ресурсів та задоволення потреб фізичниї та юридичних осіб в інформаційних послугах.

Як показує досвід США, зусилля, направлені на впорядкування електронних інформаційних ресурсів шляхом їх каталогізації дозволяє досягти інтеграції останніх до потужної системи національних бібліотечних каталогів. Через це більшість зусиль до каталогізації прикладають саме бібліотеки. Слід виділити такі проекти як InterCAT, CORC, INFOMINE, CATRIONA.

OCLC (Online Computer Library Center) — це міжнародна організація, яка включає в себе більш ніж 53 тисячі бібліотек у 96 країнах і ставить перед собою мету допомогти бібліотекам та іншим закладам організувати інформаційні ресурси таким чином, щоб шукачі знання могли знайти їх, прикладаючи мінімальні зусилля, задля створення нового знання. Для цього OCLC допомагає бібліотекам обмінюватися своїми колекці та створює рамки співпраці для організації нового знання. Основними напрямками діяльності цієї організації є: Cataloging and Metadata, Collection Management, Digitization and Preservation, Electronic content, Reference, Resource Sharing, DUNS (Data Universal Numbering System).

Більшість країн на урядовому рівні займаються розробкою стандартів для створення, розвитку та використання інформаційних ресурсів:

1. Великобританія: урядовий шлюз, урядовий Інтернет, e-GIF, e-GMS (e-Government Metadata) — стандарт, який задає елементи для опису метаданих та їх відображення Dublin Core, AGLS, NGDF, GILS та PRO.
2. Данія: інфраструктура Infostructurebase;
3. Швеція: Government Elink (Ge).
4. Австралія: FedLink — урядовий шлюз і захищений урядовий Інтранет.
5. Гонконг: Government System Architecture (GSA) і Electronic Service Delivery (ESD) Scheme.
6. США: Федеральна корпоративна архітектура інформаційних технологій державних організацій.

В наші дні значних результатів досягнуто у галузі доступу до інформаційних ресурсів за допомогою веб-сервісів.

Веб-сервіси забезпечують можливість будувати найперспективнішу сервісно-орієнтовану архітектуру (Service Oriented Architecture, SOA) розподілених інформаційних інфраструктур із використанням технологій веб-сервісів на основі промислових продуктів (наприклад, BizTalk Server 2004 від Microsoft). Очікується, що з 2007–2008 рр. альтернатив підходу SOA не буде.

02.02.2005 р. Асоціація OASIS затвердила як стандарт специфікацію UDDI 3.0, що регламентує реєстри доступних для виклику веб-сервісів. На думку експертів, UDDI 3.0 — перша версія специфікації, яка відповідає вимогам сервіс-орієнтованої архітектури. Стандарт дозволяє створювати розподілені реєстри інформаційних ресурсів. Головною його особливістю є наявність механізму підтвердження автентичності інформації у реєстрі, реалізованого на основі цифрових підписів формату XML Digital Signatures. Модель безпеки, яка використовується в UDDI 3.0, побудована на політиках — за їх допомогою контролюються керування доступом, тиражування, підписка, делегування прав на пересилання даних і ключі UDDI — унікальні ідентифікатори, які присвоюються кожному запису реєстру. UDDI 3.0 дозволяє створювати закриті, напівзакриті й відкриті реєстри з можливістю їхньої інтеграції на різних рівнях.

2.3 Висновки до розділу

У цьому розділі було розглянуто поняття електронного інформаційного ресурсу та реєстру таких ресурсів та проведено огляд існуючих рішень поставленої задачі.

3 ЗАСОБИ РОЗРОБКИ

Важливе місце, при розробці програмного продукту, займає вибір технологій та засобів щодо реалізації. Це безпосередньо впливає на такі аспекти як час, затрачений на розробку програмного продукту, його надійність, масштабованість, матеріальні затрати на подальшу підтримку продукту. Основним середовищем розробки було вибрано редактор Visual Studio Code.

Для розробки алгоритмів завантаження, вивантаження та зберігання інформаційних ресурсів було використано мову програмування Javascript, як на клієнті, так і на сервері за допомогою середовища виконання Node.js.

Для створення графічного інтерфейсу, основним інструментом було вибрано фреймворк Vue.js та надбудову над мовою Javascript – Typescript.

3.1 Редактор Visual Studio Code v1.34

Редактор Visual Studio Code – один з найпопулярніших на сьогодні інструментів для створення сучасних веб-застосунків і програм для хмарних систем. Visual Studio Code є повністю безкоштовним та доступним для таких ОС як Windows, Linux та OS X.

Цей редактор був представлений компанією Microsoft у квітні 2015 року. Він став перший крос-платформовим продуктом в лінійці Visual Studio.

Редактор має безліч вбудованих інструментів, серед яких відлагоджувач, інструменти для роботи з Git, засоби навігації по коду, автодоповнення, контекстні підказки та підтримує мову програмування Typescript без необхідності встановлення додаткових плагінів.

Visual Studio Code включає в себе редактор коду, що підтримує IntelliSense (компонент завершення коду), а також рефакторинг коду. Інтегрований відладчик працює як відладчик вихідного рівня, так і відладчик на рівні машини. Інші вбудовані інструменти включають в себе профайлер коду, дизайнер форм для створення GUI-додатків, веб-дизайнер, дизайнер класів і дизайнер схеми бази даних.

Він приймає плагіни, що підвищують функціональність практично на кожному рівні, включаючи додавання підтримки систем керування джерелами (наприклад, Subversion і Git) і додавання нових наборів інструментів, таких як редактори та візуальні дизайнери для доменних мов або наборів інструментів для інших аспектів розробки програмного забезпечення Життєвий цикл (наприклад, клієнт Team Foundation Server: Team Explorer).

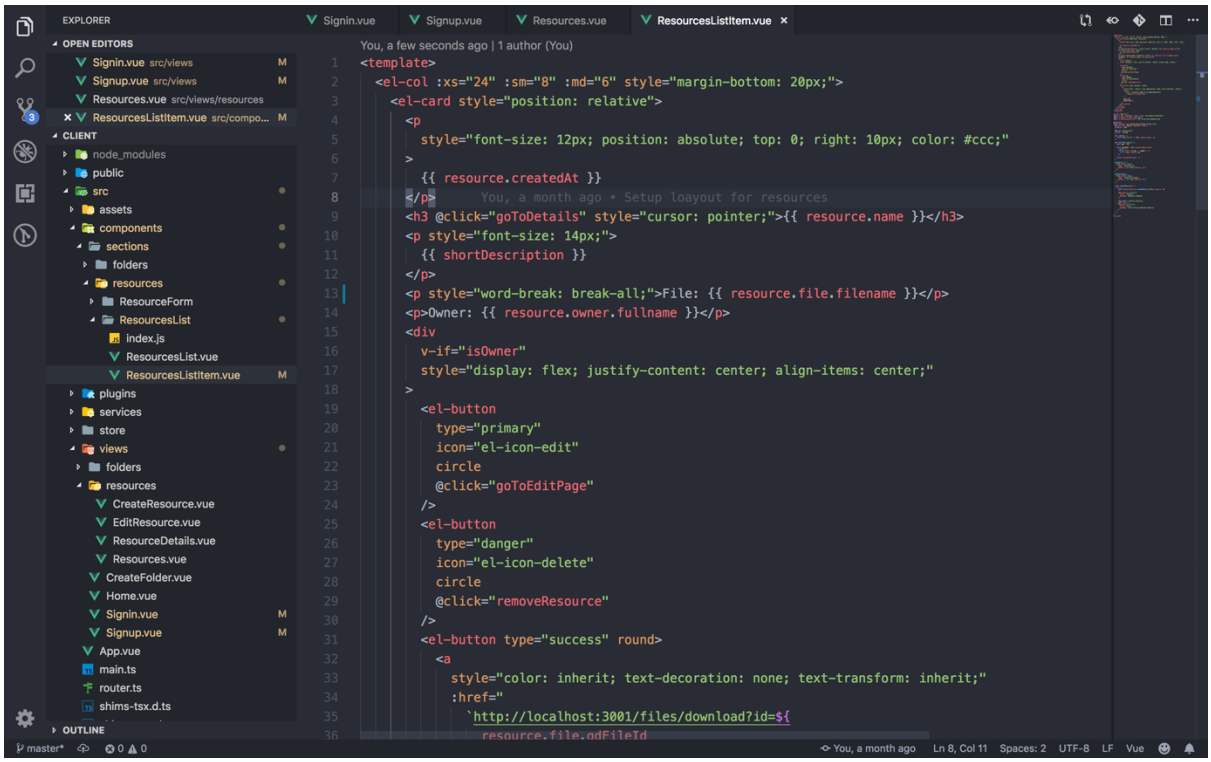

Рисунок 3.1 — Редактор Visual Studio Code

3.2 Середовище виконання Node.js

Середовище виконання Node.js – продукт з відкритим кодом, яка дозволяє запускати та виконувати програми, написані мовою програмування Javascript.

Платформа була представлена світові в травні 2009 року. засновником її є Раян Дал [3].

Якщо раніше Javascript застосовувався лише в браузері, то Node.js дало змогу використуввати Javascript і на сервері та відправляти користувачеві результати його виконання. Це дозволило розробникам використовувати одно мову програмування, як для написання клієнтських застосунків, так і для створення повноцінних

серверних застосунків. Такий підхід значно скорочує як матеріальні, так і часові витрати на розробку систем.

Node.js має наступні головні властивості:

- Однопотокова асинхронна модель виконання запитів.
- Неблокуючий ввід/вивід.
- Рушій Javascript Google V8.
- Система модулів SystemJS

Node.js працює на циклі подій одного потоку, використовуючи неблокуючі виклики вводу/виводу, що дозволяє йому підтримувати десятки тисяч одночасних з'єднань без перенесення витрат на перемикання контекстних потоків [7]. Конструкція спільного використання одного потоку серед усіх запитів, які використовують шаблон спостерігача, призначена для побудови високопаралельних додатків, де будь-яка функція, що виконує ввід/вивід, повинна використовувати зворотний виклик. Щоб розмістити однопоточний цикл подій, Node.js використовує бібліотеку libuv, яка, у свою чергу, використовує пул потоків фіксованого розміру, який обробляє деякі неблокуючі асинхронні операції вводу/виводу.

Пул потоків обробляє виконання паралельних завдань у Node.js. Основний виклик функції потоку повідомляє завдання в спільну чергу завдань, потоки якої в пулі потоків тягнуть і виконують. По суті, неблокуючі системні функції, такі як мережа, переводять на неблокуючі сокети на стороні ядра, при цьому блокуючи функції системи, такі як файловий ввід/вивід, запускають блокуючим способом на власних потоках [9]. Коли потік у пулі потоків завершує завдання, він інформує про це головний потік, який, у свою чергу, прокидається і виконує зареєстрований зворотний виклик.

Недоліком цього однопоточного підходу є те, що Node.js не дозволяє здійснювати вертикальне масштабування, збільшуючи кількість ядер процесора машини, на якому він працює, без використання додаткового модуля, наприклад кластера, або pm2. Проте розробники можуть збільшити кількість потоків за замовчуванням у пулі потоку libuv. Серверна операційна система (ОС), ймовірно, буде поширювати ці потоки в декількох ядрах. Інша проблема полягає в тому, що

довготривалі обчислення та інші завдання, пов'язані з процесором, заморожують весь цикл подій до завершення.

Головні переваги які надає використання Node.js як основної технології для створення серверних рішень, наступні:

- Серверне рішення. Node.js створює неблокувальні додатки вводу-виводу, які можуть легко обслуговувати кілька одночасних подій. Розробник може легко створити масштабоване серверне рішення, яке максимізує використання одного процесора.
- Єдина мова. Одна з найкращих частин Node.js полягає в тому, що вона використовує єдину мову через стек розробки додатків. Це є сприятливим, оскільки використання однієї мови як на передній, так і на задній панелях лише покращить функціональність вашої програми.
- Гнучка. Node.js - одна з найбільш гнучких платформ. Є дуже мало вказівок і залежностей під час використання Node.js. Немає жорстких правил під час використання Node.js.
- Складність. Node.js не надто складний для використання. Але вона все ще вимагає більше ліній кодування. Вона також вимагає від розробника розуміння функцій закриття та зворотного виклику.
- Використання JSON. PHP не використовує JSON стільки, скільки Node.js. PHP використовує головним чином функції json_encode [] і json_decode []. JSON працюватиме краще у випадку Node.js.
- Швидкість виконання. PHP набагато повільніше порівняно з Node.js. Хоча Node.js не просто швидше, ніж PHP, але він також легкий.
- Веб-сервер. PHP вимагає запуску веб-сервера Apache. Тоді як Node.js не потрібен веб-сервер. Він працюватиме у власному середовищі виконання.

3.3 Мова програмування Javascript

Мова програмування Javascript – це мультипарадигменна мова програмування. Яка вдало поєднує в собі можливості для написання коду як в об’єктно-

орієнтованому, так и в імперативному чи функціональному стилях. Є реалізацією специфікації ECMAScript [8].

У листопаді 1996 року Netscape представив JavaScript для ECMA International, щоб отримати стандартну специфікацію, яку інші виробники браузерів могли б реалізувати на основі роботи, виконаної в Netscape. Це призвело до офіційного випуску мовної специфікації ECMAScript, опублікованої в першому виданні стандарту ECMA-262 у червні 1997 року, причому JavaScript є найбільш відомим з реалізації. ActionScript і JScript були іншими відомими реалізаціями ECMAScript.

Випуск ECMAScript 2 у червні 1998 року продовжив цикл процесів стандартів, відповідно до деяких змін до міжнародного стандарту ISO / IEC 16262. ECMAScript 3 вийшов у грудні 1999 року і є сучасною базою для JavaScript. Оригінальна робота ECMAScript 4 під керівництвом Вальдемара Хорвата (тоді в Netscape, тепер у Google) почалася в 2000 році. Microsoft спочатку брав участь і впроваджував деякі пропозиції на своїй мові JScript .NET.

З часом стало зрозуміло, що Microsoft не має наміру співпрацювати або реалізовувати належний JavaScript в Internet Explorer, навіть якщо вони не мали конкуруючих пропозицій, і вони мали часткову (і відхилялася) реалізацію на сервері .NET. Таким чином, до 2003 року початкова робота ECMAScript 4 була завершена.

Наступна велика подія відбулася у 2005 році з двома основними подіями в історії JavaScript. По-перше, Брендан Ейч і Mozilla знову приєдналися до Ecma International як неприбутковий член і розпочали роботу над ECMAScript для XML (E4X), стандарту ECMA-357, який надійшов від колишніх співробітників Microsoft у BEA Systems (спочатку придбаний як Crossgain) . Це призвело до спільної роботи з Macromedia (пізніше придбаної компанією Adobe Systems), яка впроваджувала E4X в ActionScript 3 (ActionScript 3 був вилкою оригінального ECMAScript 4).

Таким чином, разом з Macromedia, робота перезавантажена на ECMAScript 4 з метою стандартизації того, що було в ActionScript 3. Для цього Adobe Systems випустила віртуальну машину ActionScript 2, коду на ім'я Tamarin, як проект з відкритим кодом. Але Tamarin і ActionScript 3 занадто відрізнялися від веб-

JavaScript для зближення, як це було реалізовано сторонами в 2007 і 2008 роках.

На жаль, між різними гравцями все ще були потрясіння; Дуглас Крокфорд (Douglas Crockford) - тоді в Yahoo! - об'єднав сили з Microsoft в 2007 році, щоб протиставити ECMAScript 4, що призвело до зусиль ECMAScript 3.1. Розробка ECMAScript 4 ніколи не була завершена, але ця робота вплинула на наступні версії.

Хоча все це відбувалося, спільноти з відкритим кодом і розробниками працювали над тим, щоб революціонізувати те, що можна зробити за допомогою JavaScript. Це зусилля спільноти було розпочато у 2005 році, коли Джессі Джеймс Гарретт випустив білу книгу, в якій він винайшов термін Ajax, і описав набір технологій, з яких JavaScript був основою, що використовується для створення веб-додатків, де дані можуть бути завантажені у фоновому режимі , що дозволяє уникнути необхідності перезавантаження сторінок і призводить до більш динамічних програм. Це призвело до ренесансного періоду використання JavaScript, очолюваного бібліотеками з відкритим вихідним кодом та спільнотами, які формувалися навколо них, з бібліотеками, такими як Prototype, jQuery, Dojo Toolkit, MooTools та інші.

У липні 2008 року в Осло зібралися різні партії з обох сторін. Це призвело до можливої домовленості на початку 2009 року про перейменування ECMAScript 3.1 на ECMAScript 5 і просування мови вперед, використовуючи порядок денний, який відомий як Гармонія. ECMAScript 5 був нарешті випущений в грудні 2009 року.

У червні 2011 року ECMAScript 5.1 був випущений для повного узгодження з третім виданням міжнародного стандарту ISO / IEC 16262. ECMAScript 2015 вийшов у червні 2015 року. ECMAScript 2016 вийшов у червні 2016 року. Поточна версія ECMAScript 2017, випущена у червні 2017 року.

Представлений світові Javascript був в 1995 році. Автором його став Брендан Ейх, який в той час працював розробников в компанії Netscape. Основна ціль при створенні цього інструменту була забезпечити «мову для склеювання» зіставних частин веб-застосунку: палгінів Java-аплетів, який мав бути зручним для веб-дизайнерів та програмістів не володіючих високою кваліфікацією.

До основних архітектурних рис Javascript відносять:

- динамічна типізація;
- слабка типізація;
- прототипне програмування;
- автоматичне керування пам’яттю;
- функції, як об’єкти першого класу.

Структурно мову можливо представити, як об’єднання трьох чітко розмежованих одна від одної частин:

- ядро (ECMAScript);
- об’єктна модель браузера (Browser Object Model);
- об’єктна модель документа (Document Object Model).

Ядро – специфікація ECMAScript не є мовою програмування та в ній не є визначені методи для вводу/виводу інформації. Це є основа для побудови скриптових мов програмування. Специфікація ECMAScript описує типи данних, інструкції, ключові та зарезервовані слова, оператори, об’єкти, не обмежуючи авторів мов програмування від розширення їх.

Об’єктна модель браузера – частина мови яка є специфічною лише для випадку, коли середовищем виконання Javascript є браузер. Являє собою деяку проміжну ланку між ядром та об’єктною моделлю документа. Основне призначення її – керування вікнами браузера та забезпечення їх взаємодії

Об’єктна модель документа – інтерфейс програмування застосунків для HTML та XML документів. Згідно цієї моделі, документ (наприклад веб-сторінка) може бути представлена у вигляді дерева об’єктів, які мають певні властивості і дозволяють проводити з ними наступні дії:

- генерація та додавання вузлів;
- отримання вузлів;
- зміна вузлів;
- зміна зв’язків між вузлами;

- видалення вузлів.

3.4 Мова програмування TypeScript

Мова програмування TypeScript – мова програмування, представлений компанією Microsoft в 2012 році. Позиціонується як засіб для розробки веб-застосунків, який розширює можливості Javascript.

TypeScript походить від недоліків JavaScript для розробки великомасштабних програм, як у Microsoft, так і серед їхніх зовнішніх клієнтів. Проблеми зі складним кодом JavaScript призвели до потреби в користувальницьких інструментах для полегшення розробки компонентів мовою.

Розробники TypeScript шукали рішення, яке б не порушувало сумісність зі стандартом і його підтримкою між платформами. Знаючи, що поточна стандартна пропозиція ECMAScript обіцяє майбутню підтримку для програмування на основі класів, TypeScript базувався на цій пропозиції. Це призвело до компілятора JavaScript з набором розширень синтаксичних мов, розширених на основі пропозиції, що перетворює розширення у звичайний JavaScript. У цьому сенсі TypeScript був попереднім переглядом того, чого очікувати від ECMAScript 2015. Унікальний аспект, який не входить до пропозиції, але додається до TypeScript, є додатковим статичним типізацією, що дозволяє статичний аналіз мови, що полегшує інструменти та підтримку IDE.

TypeScript є повністю сумісний з Javascript та компілюється в нього. Тобто, будь яких код, написаний на TypeScript може бути скомпільовано в Javascript та запускатися в тих самих оточеннях.

Відрізняється TypeScript від Javascript тим, що він додає до останнього строгу типізацію, що дозволяє уникнути безлічі помилок ще на етапі написання коду, підтримку класового синтаксису, як в класичних об’єктно-орієнтованих мовах програмування, підтримкою модулів, що здатно підвищити швидкість розробки, полегшити читаємість, рефакторінг, та повторне використання коду.

Загалом, TypeScript додає до специфікації ECMAScript 5 такі опції:

- Анотації типів та їх перевірка на етапі компіляції;
- вивід типів;
- класи;
- інтерфейси;
- перераховувані типи (enum);
- модулі;
- додаткові параметри та параметри за замовчуванням;
- кортежі.

3.5 Фреймворк Vue.js

Фреймворк Vue.js – Javascript-фреймворк, що використовує шаблон MVVM для створення інтерфейсів користувача на основі моделей даних, через реактивне зв'язування даних.

Даний іструмент був створений, на той час, працівником компанії Google Creative Labs – Еваном Ю. Працюючи над одним з проектів йому довелось швидко будувати прототип складого користувацього інтерфейсу. Так як інструментів для цього на той час ще не було, він створив власний фреймворк.

До головних особливостей Vue.js відносять:

- Шаблони;
- реактивність;
- компоненти;
- анімаційні переходи;
- роутинг.

Для опису користувацього інтерфейсу Vue.js використовує синтаксис базований на HTML. Будь-який Vue.js шаблон є валідним HTML кодом, зрозумілим браузеру. Всередині такий шаблон представляється у вигляді Javascript-моделі – рендер-функції. Завдяки цьому, і системі рективності Vue.js розробник може створювати динамічні інтерфейси неймвірно швидко та просто.

Система реактивності Vue.js побудована на такії можливості мови

програмування Javascript як Object.prototype.defineProperty, яка дозволяє встановлювати на змінні геттери та сеттери і таким чином відсліжувати їх зміни та змінювати шаблони відповідно до них.

Vue.js реалізує компонентний підхід щодо створення користувацьких інтерфейсів. Основний спосіб опису Vue.js компонента є SFC (Single File Component) або «Однофайловий компонент». Це дозволяє швидко переходити від написання HTML-розмітки до Javascript і навпаки, так як і те, і інше знаходиться в одному файлі.

Для надання користувацьким інтерфейсам більш «живиго» та привабливого вигляду, в веб-розробці використовують анімації та переходи. Vue.js має зручну та потужну вбудовану систему для їх реалізації.

Більшість застосунків, створених за допомогою Vue.js – це Single Page Applications (SPA). Відрізняються вони від традиційних сайтів тим, що переходи між сторінками та будь-які дії користувача не викликають перезавантаження сторінки в браузері. Це призводить до покращення користувацького досвіду [6]. Vue.js має вбудуваний зручний інструмент для роботи з браузерною навігацією vue-router.

SPA - це веб-додаток або веб-сайт, який взаємодіє з користувачем, динамічно переписуючи поточну сторінку, а не завантажуючи нові сторінки з сервера [1]. Цей підхід дозволяє уникнути переривання користувацького досвіду між послідовними сторінками, роблячи програму більш схожою на настільну програму. У SPA, весь необхідний код - HTML, JavaScript, і CSS - витягується з одного завантаження сторінки, або відповідні ресурси динамічно завантажуються і додаються на сторінку, як це необхідно, як правило, у відповідь на дії користувача. Сторінка не перезавантажується в будь-якій точці процесу, а також не здійснює керування передачею на іншу сторінку, хоча хеш розташування або API історії HTML5 можна використовувати для забезпечення сприйняття та навігації окремих логічних сторінок програми. Взаємодія з програмою для однієї сторінки часто передбачає динамічну комунікацію з веб-сервером за кадром.

Серед основних переваг використання SPA перед використанням традиційного підходу MPA (Multi-page Application) виділяють:

- SPA є швидким, оскільки більшість ресурсів (HTML + CSS + Scripts) завантажуються лише один раз протягом всього життя програми. Тільки дані передаються вперед і назад.
- Спрощено процес розробки. Немає необхідності писати код для візуалізації сторінок на сервері. Початок роботи набагато простіше, тому що ви можете запустити розробку з файлу: // URI, не використовуючи жодного сервера.
- SPA легко піддаються налагодженню за допомогою Chrome, оскільки ви можете контролювати мережеві операції, досліджувати елементи сторінки та дані, пов'язані з нею.
- Легше зробити мобільний додаток, оскільки розробник може повторно використовувати той же код для веб-застосунків і рідного мобільного додатка.
- SPA може ефективно кешувати будь-яке локальне зберігання. Додаток надсилає тільки один запит, зберігає всі дані, потім може використовувати ці дані і працює навіть в автономному режимі.

3.6 Хмарне сховище Google Drive

Хмарне сховище Google Drive - це служба зберігання та синхронізації файлів, розроблена Google. Запущений 24 квітня 2012 року, Google Drive дозволяє користувачам зберігати файли на своїх серверах, синхронізувати файли між пристроями та обмінюватися файлами [2]. Окрім веб-сайту, Диск Google пропонує програми з функціями, які не входять до мережі, для комп'ютерів Windows і MacOS, а також смартфонів і планшетів Android і iOS. Диск Google включає Документи Google, Таблиці Google і слайди Google, які є частиною офісного пакету, який дозволяє спільне редагування документів, електронних таблиць, презентацій, малюнків, форм тощо. Файли, створені та відредаговані через офісний пакет, зберігаються на Диску Google.

Диск Google пропонує користувачам 15 гігабайт вільного сховища через Google One. Google One також пропонує 100 гігабайт, 200 гігабайт, 2 терабайти, 10

терабайт, 20 терабайт і 30 терабайт, пропоновані через додаткові платні плани. Розмір завантажених файлів може становити до 5 терабайт. Користувачі можуть змінювати параметри конфіденційності для окремих файлів і папок, включаючи можливість спільного використання з іншими користувачами або публікувати вміст. На веб-сайті користувачі можуть шукати зображення, описуючи його візуальні ефекти, і використовувати природну мову для пошуку конкретних файлів, таких як "знайти бюджетну таблицю з минулого грудня".

Веб-сайт і додаток для Android пропонують розділ резервного копіювання, щоб дізнатися, які пристрої Android підтримують дані служби, а повністю оновлений комп'ютерний додаток, випущений у липні 2017 року, дозволяє створювати резервні копії певних папок на комп'ютері користувача. Функція швидкого доступу може розумно передбачити, які файли потрібні користувачам.

Диск Google є ключовим компонентом GSuite, щомісячною пропозицією Google для компаній і організацій. Як частина планів GSuite, Goolge Drive надає необмежену кількість зберігань, розширену звітність з аудиту файлів, вдосконалені засоби адміністрування та більше інструментів співпраці для команд.

Після запуску сервісу політика конфіденційності Google Drive була піддана великій критиці з боку деяких представників ЗМІ. У Google є один набір угод про умови обслуговування та політику конфіденційності, які охоплюють всі його послуги, а це означає, що мова в угодах надає компанії широкі права на відтворення, використання та створення похідних творів із вмісту, збереженого на Google Drive. Хоча політика також підтверджує, що користувачі зберігають права інтелектуальної власності, захисники конфіденційності висловлюють занепокоєння, що ліцензії надають Google права використовувати інформацію та дані для налаштування реклами та інших послуг, які надає Google. Навпаки, інші представники ЗМІ зазначили, що угоди не гірші, ніж у конкуруючих послуг з зберігання даних у хмарі, але що конкуренція використовує "більш хитру мову" у угодах, а також заявила, що Google потребує прав для того, щоб "рухатися" файли на своїх серверах, кешувати дані або створювати ескізи зображень ".

Диск Google має веб-сайт, який дозволяє користувачам переглядати свої

файли з будь-якого комп'ютера, підключеного до Інтернету, без необхідності завантаження програми.

Веб-сайт отримав візуальну реконструкцію у 2014 році, що дало йому зовсім новий вигляд і поліпшену продуктивність. Він також спростив деякі з найбільш поширених завдань, таких як натискання лише одного разу на файл, щоб побачити нещодавню активність або спільний доступ до файлу, а також додану функцію перетягування, де користувачі можуть просто перетягувати вибрані файли до папок для покращення організації.

Нове оновлення у серпні 2016 року змінило кілька візуальних елементів веб-сайту; оновлено логотип, оновлено вікно пошуку, а основний колір змінено з червоного на синій. Також було покращено функціональність для завантаження файлів з веб-сайту; Користувачі тепер можуть стискати та завантажувати великі елементи диска в декілька файлів у форматі .zip з покращеною структурою іменування, краще керування формами Google, а тепер пусті папки будуть включені в .zip, таким чином зберігаючи ієрархію папок користувача.

3.7 База даних MongoDB

MongoDB - відкрите програмне забезпечення, яке є розширюваною, високопродуктивною, вільною від схем, документо-орієнтованою базою даних, яка написана на C++. Розробляється з жовтня 2007 року компанією 10gen. Зберігає всі ваші дані у форматі бінарного JSON (BSON). Вже має широке застосування у реальних проектах.

Основні можливості MongoDB:

- Документо-орієнтоване сховище (проста та потужна JSON-подібна схема даних).
- Повна підтримка індексів.
- Підтримка відмовостійкості і масштабованості.
- Гнучка мова для формування запитів.
- Динамічні запити.
- Профілювання запитів.

- Швидкі оновлення.
- Журналювання операцій, що модифікують дані в БД.
- Підтримка MapReduce.

Нереляційні бази даних (або NoSQL) охоплюють широкий спектр різних технологій баз даних, які були розроблені у відповідь на вимоги, що пред'являються при побудові сучасних додатків:

- Розробники працюють з додатками, які створюють великі обсяги нових, швидко мінливих типів даних - структуровані, напівструктуровані, неструктуровані і поліморфні дані.
- Довгий минулий цикл розвитку водоспаду дванадцять-вісімнадцять місяців. Тепер невеликі команди працюють у спринтах, швидко перебирають коди кожний тиждень або два, деякі навіть кілька разів на день.
- Програми, які колись обслуговувалися обмеженою аудиторією, тепер поставляються як послуги, які повинні бути постійними, доступними для багатьох різних пристроїв і масштабуватися у мільйонах користувачів у глобальному масштабі.
- В даний час організації звертаються до масштабних архітектур з використанням відкритих програмних технологій, товарних серверів і хмарних обчислень замість великих монолітних серверів і інфраструктури зберігання даних.

У деяких сценаріях невдач, коли програма може отримати доступ до двох різних процесів MongoDB, але ці процеси не можуть отримати доступ один до одного, MongoDB може повернути застарілі читання. У цьому сценарії також можливо, щоб MongoDB відкинув записи, які були визнані. Ця проблема була розглянута після версії 3.4.0, випущеної в листопаді 2016 року (і повернута до v3.2.12).

До версії 2.2, замки були реалізовані на кожному сервері. У версії 2.2 замки були реалізовані на рівні бази даних. Починаючи з версії 3.0 були введені двигуни запам'ятовуючого пристрою для зберігання даних, і кожен двигун зберігання може реалізовувати замки по-різному. За допомогою MongoDB 3.0 замки реалізуються на рівні збору для механізму зберігання MMAPv1, а механізм зберігання WiredTiger

використовує оптимістичний протокол паралелізму, який ефективно забезпечує блокування на рівні документа. Навіть з версіями до 3.0, одним з підходів до збільшення паралельності є використання sharding. У деяких ситуаціях читає та записує їхні замки. Якщо MongoDB передбачає, що сторінка навряд чи перебуватиме в пам'яті, операції будуть приводити до блокування під час завантаження сторінок. Використання блокувального розширення значно розширилося в 2.2.

До версії 3.3.11, MongoDB не міг робити сортування на основі сортування і обмежувалося порівнянням байтів через memcmp, який не забезпечував правильного упорядкування для багатьох не англійських мов при використанні кодування Unicode. Питання було виправлено 23 серпня 2016 року.

За допомогою механізму зберігання MMAPv1 запити MongoDB щодо індексу не є атомними і можуть пропускати документи, які оновлюються під час виконання запиту, і відповідають запиту до і після оновлення. До MongoDB 4.0 запити щодо індексу не були атомними і могли пропустити документи, які оновлювалися під час виконання запиту, і відповідали запиту до і після оновлення. Введення знімка, що читається, занепокоєння в MongoDB 4.0 усунуло це явище.

MongoDB є нереляційною базою даних. Переваги нереляційних баз даних над реляційними наступні:

- Вони масштабуються горизонтально і працюють з неструктурованими та напівструктурованими даними. Деякі підтримують послідовність транзакцій ACID.
- Висока доступність.
- У той час як багато баз даних NoSQL є відкритими та безкоштовними, часто існують значні витрати на навчання, налаштування та розробки. Тепер також доступні численні комерційні продукти.

3.8 Метамова SASS

Метамова Sass (Syntactically awesome style sheets) - це мова стилю, спочатку розроблена Хемптоном Кетліном і розроблена Наталі Вейзенбаум. Після своїх

початкових версій, Weizenbaum і Chris Eppstein продовжують розширювати Sass з SassScript, простою мовою сценаріїв, що використовується у файлах Sass.

Sass - це мова сценаріїв препроцесора, яка інтерпретується або компілюється в каскадні таблиці стилів (CSS). SassScript - сама мова сценаріїв. Sass складається з двох синтаксисів. Оригінальний синтаксис, який називається "синтаксис з відступом", використовує синтаксис, подібний до Haml. Він використовує відступи для розділення кодів і символів нового рядка для розділення правил. Новий синтаксис "SCSS" (Sassy CSS) використовує блокове форматування, подібне до CSS. Він використовує фігурні дужки для позначення кодів і точок з комою для окремих рядків у межах блоку.

CSS3 складається з серії селекторів і псевдоселекторів, які групують правила, що застосовуються до них. Sass (у більш широкому контексті обох синтаксисів) розширює CSS, надаючи кілька механізмів, доступних у більш традиційних мовах програмування, зокрема об'єктно-орієнтованих мовах, але які не доступні самому CSS3. Коли SassScript інтерпретується, він створює блоки правил CSS для різних селекторів, як це визначено у файлі Sass. Інтерпретатор Sass переводить SassScript у CSS. Крім того, Sass може контролювати файл .sass або .scss і переводити його у вихідний файл .css кожного разу, коли зберігається файл .sass або .scss.

Відступний синтаксис є метамовою. SCSS є вкладеним метамовою, так як допустимий CSS є дійсним SCSS з тією ж семантикою.

SassScript надає такі механізми: змінні, вкладеність, змішування та успадкування селектора.

3.9 Вимоги до системи

Для корректої роботи системи рекомендовано використовувати браузери Google Chrome 6.0+, Opera 9.7+, Internet Explorer 9.0+, Safari 3.1+ або Mozilla FireFox 4.0+.

3.10 Висновки до розділу

В цьому розділі були розглянуті засоби, які були використані для розробки

сервісу. Так як розроблена система є web-сервісом і призначена для роботи з нею через браузер, було обрано єдиний варіант мови розмітки HTML, каскадні стилі CSS та мова програмування Javascript для клієнтської сторони. Для серверної частини було також обрано Javascript, як мову програмування. З цією метою було використано середовище виконання Node.js. Таке рішення було зумовлено тим, що використання однією мови програмування для клієнта та для сервера, допомагає значно заощадити час на розробку, так як не потрібно вивчати ще одну мову програмування та скоротити матеріальні затрати на подальшу підтримку продукту. Для клієнтської частини, також було використано фреймворк Vue.js, який є одним з найпопулярніших останнім часом Javascript-фрейморком. Також для клієнтської частини застосунку було вирішено використовувати мову програмування TypeScript, що дозволило будувати систему опираючись на строгу типізацію та принципи об’єктно-орієнтованого програмування.

4 ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

В цьому розділі ми розглянемо опис розробленого програмного продукту та особливості програмної реалізації модулів розробленої системи:

- Серверна частина;
- клієнтська частина;
- база даних.

4.1 Загальний опис системи

В системі можна виділити два основних типа користувачів:

- Звичайний користувач;
- адміністратор системи.

Для кожної ролі визначено певний перелік доступних дій. Без авторизації система є недоступною для користувача. Після авторизації в системі, звичайний користувач може:

- Переглядати список існуючих інформаційних ресурсів;
- створювати інформаційні ресурси;
- редагувати інформаційні ресурси;
- вивантажувати інформаційні ресурси на локальний комп’ютер.

Адміністратор, в свою чергу, має такі можливості:

- Переглядати список зареєстрованих користувачів;
- обмежувати доступ до ресурсу для певного користувача;
- модерувати існуючі інформаційні ресурси.

Докладніше можливості користувачів можна побачити на діаграмі прецедентів (Рисунок 4.1).

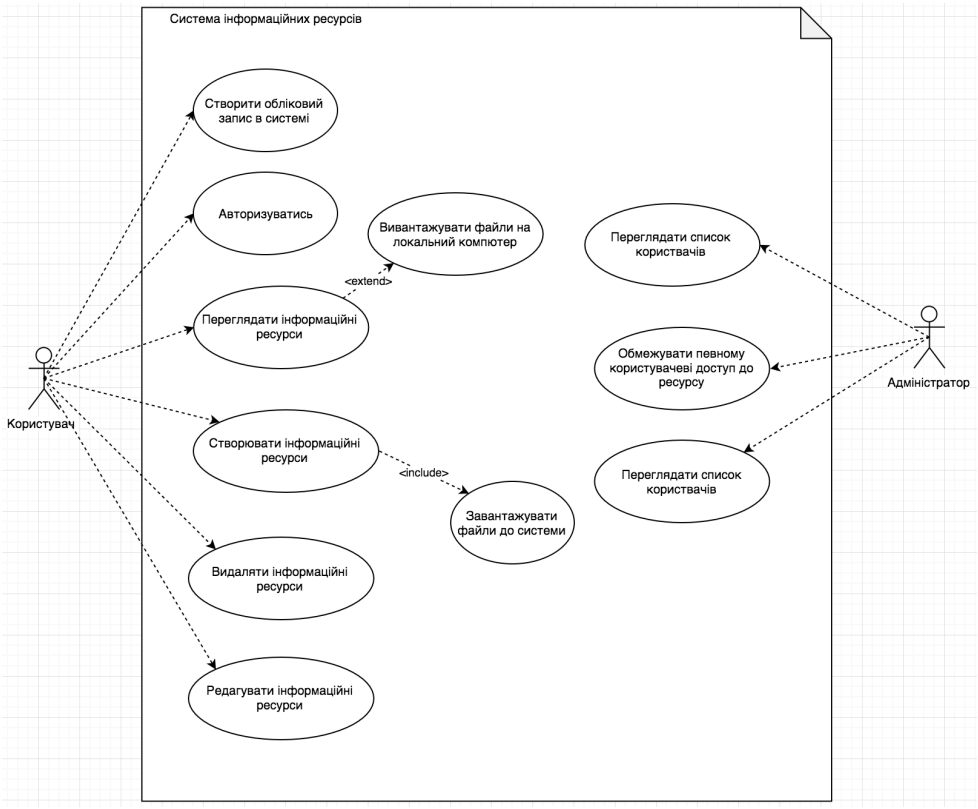

Рисунок 4.1 - Діаграма прецедентів системи

4.2 Архітектура серверної частини

Так як серверна частина виступає в ролі API (Application programming interface, програмний інтерфейс застосунку) для клієнтського застосунку, то для її реалізації було використано підхід REST (Representational State Transfer, передача репрезентативного стану) [4].

Використовуючи протокол без статусу та стандартні операції, RESTful системи прагнуть до швидкої продуктивності, надійності та здатності рости за допомогою повторного використання компонентів, які можна керувати та оновлювати, не впливаючи на систему в цілому, навіть під час її роботи.

Термін REST був введений і визначений Рой Філдінгом у 2000 році в його докторській дисертації. Дисертація Філдінга пояснила принципи REST, які були відомі як "об'єктна модель HTTP", починаючи з 1994 року, і були використані при розробці стандартів HTTP 1.1 та Uniform Resource Identifiers (URI). Термін призначений для того, щоб викликати образ того, як веде себе добре розроблене веб-додаток: це мережа веб-ресурсів (віртуальна стан-машина), де користувач

просувається через додаток, вибираючи ідентифікатори ресурсів, такі як операції з ресурсами, такі як GET або POST (переходи стану програми), що призводить до передачі кінцевого користувача наступному поданню ресурсу (наступного стану програми) для їх використання.

Підхід до проектування API REST - це підхід до архітектури мережевих протоколів, які забезпечують доступ до інформаційних ресурсів. Був описаний і популяризований 2000 року Роєм філдінгом, одним із творців протоколу HTTP. В основі REST закладено принципи функціонування всесвітньої павутини і, зокрема, можливості HTTP. Філдінг розробив REST паралельно з HTTP 1.1 базуючись на попередньому протоколі HTTP 1.0.

Як і будь-який архітектурний стиль REST накладає ряд архітектурних обмежень для програмного продукту, серед них:

- Вимога до розділення відповідальності між клієнтом і сервером;
- відсутність стану - кожний запит не здійснює ніякий побічних ефектів і є повністю ізольованим;
- системи мають підтримувати кешування.

Для реалізації зв’язку між клієнтом та сервером, архітектурний стиль REST виділяє такі основні HTTP методи:

- GET - отримання представлення ресурсу;
- POST - створення ресурсу;
- PUT - редагування ресурсу;
- DELETE - видалення ресурсу;

В вибраному фреймоворку для реалізації серверної частини, широко використовує архітектурний патерн Interceptor (перехоплювач)

Паттерн Interceptor є шаблоном розробки програмного забезпечення, який використовується, коли програмні системи або фреймворки хочуть запропонувати спосіб змінити або збільшити свій звичайний цикл обробки. Наприклад, (спрощена) типова послідовність обробки для веб-сервера полягає в тому, щоб отримати URI з браузера, зіставити його з файлом на диску, відкрити файл і надіслати його вміст до браузера. Будь-який з цих кроків може бути замінений або змінений, наприклад,

шляхом заміни шляхів URI, які відображаються в іменах файлів, або шляхом вставки нового кроку, який обробляє вміст файлів. Схема даного паттерну зображена на рисунку 4.2.

Рисунок 4.2 – Паттерн Interceptor

Ключовими аспектами цієї моделі є те, що зміна є прозорою і використовується автоматично. По суті, решта системи не повинна знати, що щось було додано або змінено і може продовжувати працювати як і раніше. Для полегшення цього має бути реалізований попередньо визначений інтерфейс для розширення, необхідний певний механізм диспетчеризації, де реєструються перехоплювачі (це може бути динамічним, під час виконання або статичним, наприклад, через конфігураційні файли) і надаються контекстні об'єкти, які дозволяють надати доступ до внутрішнього стану рамки.

Так як в серверний застосунок в основному здійснює обробку різних масивів даних. Було прийнято рішення використувувати функціональний підхід для написання коду серверної частини. Використання функціонального підходу має такі переваги:

- Функціональний підхід відомий своїми абстракціями високого рівня, які приховують велику кількість деталей таких рутинних операцій, як ітерація. Це робить код коротшим і, як наслідок, гарантує меншу кількість помилок, які можна допустити.

- У функціональному програмуванні існує менша кількість мовних примітивів. Добре відомі класи не використовуються в функціонлальному програмуванні. Замість створення унікального опису об'єкта з операціями у вигляді методів, у функціональному програмуванні є кілька основних мовних примітивів, які добре оптимізовані всередині.

- Функціонльний підхід пропонує деякі нові та цікаві інструменти для вирішення складних завдань, які розробники ООП часто нехтують.
- Робота з функціональними мовами забезпечує точне та швидке написання коду, полегшує тестування та налагодження. Ви працюєте з програмами високого рівня, а підписи функцій є більш інформативними.

Функціональний підхід заснований на таких основних концептах:

- Функції вищого порядку;
- чисті функції;
- рекурсія;
- типізація;
- прозорість посилань;

Функції вищого порядку є функціями, які можуть приймати інші функції як аргументи або повертати їх як результати. Функції вищого порядку тісно пов'язані з функціями першого класу, оскільки функції вищих порядків і функції першого класу дозволяють функції як аргументи, так і результати інших функцій. Відмінність між ними тонка: "вищий порядок" описує математичну концепцію функцій, що діють на інших функціях, тоді як "перший клас" - це комп'ютерний термін, який описує суб'єкти мови програмування, які не мають обмежень щодо їх використання (таким чином Функції першого класу можуть з'являтися в будь-якій точці програми, яку можуть мати інші об'єкти першого класу, такі як числа, включаючи як аргументи для інших функцій, так і їхні повернені значення). Функції вищого порядку дозволяють часткове застосування або викривлення, метод, який застосовує функцію до своїх аргументів по одному, при цьому кожна програма повертає нову функцію, яка приймає наступний аргумент. Це дозволяє програмісту коротко виразити, наприклад, функцію наступника як оператора додавання, частково застосованого до натурального числа.

Чисті функції (або вирази) не мають побічних ефектів (пам'ять або введення-виведення). Це означає, що чисті функції мають кілька корисних властивостей, багато з яких можна використовувати для оптимізації коду:

- Якщо результат чистого виразу не використовується, його можна видалити, не впливаючи на інші вирази.
- Якщо чиста функція викликається з аргументами, які не викликають побічних ефектів, результат є постійним відносно цього списку аргументів (іноді його називають референційною прозорістю), тобто виклик чистої функції знову з тими ж аргументами повертає той же результат. (Це може дозволити оптимізацію кешування, наприклад, запоминання.)
- Якщо між двома чистими виразами відсутня залежність від даних, їх порядок може бути скасовано, або вони можуть виконуватися паралельно, і вони не можуть втручатися один в одного (іншими словами, оцінка будь-якого чистого виразу є безпечною для потоку).
- Якщо вся мова не допускає побічних ефектів, може бути використана будь-яка стратегія оцінки; це дає компілятору можливість змінити порядок або об'єднати оцінку виразів у програмі (наприклад, з використанням знеліснення).

Хоча більшість компіляторів для імперативних мов програмування виявляють чисті функції і виконують усунення спільної підекспресії для чистих викликів функцій, вони не завжди можуть зробити це для попередньо скомпільованих бібліотек, які, як правило, не викривають цю інформацію, тим самим запобігаючи оптимізації, що стосуються цих зовнішніх функцій. Деякі компілятори, наприклад gcc, додають додаткові ключові слова для програміста, щоб явно позначити зовнішні функції як чисті, щоб дозволити такі оптимізації. Fortran 95 також дозволяє призначити функції чистими.

Ітерація (цикл) у функціональних мовах зазвичай здійснюється за допомогою рекурсії. Рекурсивні функції викликають себе, дозволяючи операції повторюватися, поки не досягнуть базового випадку. Хоча деяка рекурсія вимагає підтримки стека, рекурсія хвоста може бути розпізнана і оптимізована компілятором в той же код, який використовується для реалізації ітерації в імперативних мовах. Стандарт мови Scheme вимагає реалізації, щоб розпізнавати і оптимізувати рекурсію хвоста. Оптимізація рекурсії хвоста може бути реалізована шляхом перетворення програми на продовження стилю під час компіляції, серед інших підходів. Загальні схеми

рекурсії можна абстрагувати за допомогою функцій вищого порядку, причому найбільш очевидними прикладами є катаморфізми і анаморфізми (або "складки" і "розгортається"). Такі схеми рекурсії грають роль, аналогічну вбудованим керуючим структурам, таким як цикли на імперативних мовах. Більшість мов функціонального програмування загального призначення дозволяють необмежену рекурсію і завершуються Тьюрінгом, що робить проблему зупинки нездійсненною, може викликати нездатність міркувань рівняння, і взагалі вимагає введення неузгодженості в логіку, що виражається системою типу мови. Деякі мови спеціального призначення, такі як Coq, дозволяють лише добре обгрунтовану рекурсію і сильно нормалізуються (нетермінуючі обчислення можуть бути виражені лише з нескінченними потоками значень, які називаються кодатами). Як наслідок, ці мови не можуть бути повною, а вираження певних функцій у них неможливе, але вони все ще можуть виражати широкий клас цікавих обчислень, уникаючи при цьому проблем, що вводяться необмеженою рекурсією. Функціональне програмування з обмеженою рекурсією з деякими іншими обмеженнями називається повним функціональним програмуванням.

4.3 Архітектура клієнтської частини

Так як основним інструментом, для створення користувацького інтерфейсу, було вибрано фреймворк Vue.js який побудовано на архітектурному патерні MVVM (Model-view-viewmodel), то, при створенні клієнтської частини, було використано саме його [5].

Як архітектурний паттерн, MVVM полегшує розділ розробки графічного інтерфейсу користувача - будь то через мову розмітки або код GUI - від розробки бізнес-логіки або локальної логіки (моделі даних). View model в MVVM є перетворювачем величини, що означає, що модель перегляду відповідає за викриття (перетворення) об'єктів даних з моделі таким чином, що об'єкти легко керуються і подаються. У цьому відношенні модель перегляду є більш моделлю, ніж перегляд, і обробляє більшість, якщо не всю, логіку відображення перегляду. View model може реалізовувати шаблон медіатора, організовуючи доступ до локальної логіки навколо

набору випадків використання, підтримуваних видів. Схема взаємодії основних компонентів, які визначає даний паттерн. (Рисунок 4.3)

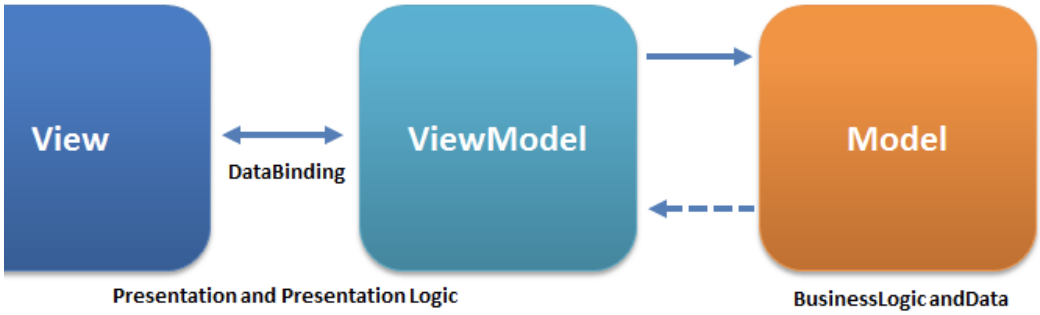

Рисунок 4.3 - Паттерн MVVM

Докладніше про основні компоненти паттерну MVVM:

- Model (Модель). Модель відноситься або до моделі домену, яка представляє вміст реального стану (об'єктно-орієнтований підхід), або до рівня доступу до даних, який представляє зміст (підхід, орієнтований на дані).
- View (Представлення). Як і в Model-View-Controller (MVC) і Model-View-Presenter (MVP), представлення - це структура, макет і вигляд того, що користувач бачить на екрані. Він відображає представлення моделі та отримує взаємодію користувача з представленням (клацання, клавіатура, жести тощо), і він пересилає обробку їх моделі представлення за допомогою прив'язки даних (властивості, зворотні виклики подій тощо). що визначається для зв'язку моделі представлення та представлення.
- View Model (Модель представлення). Модель перегляду є абстракцією перегляду, що викриває загальні властивості та команди. Замість контролера зразка MVC або презентатора шаблону MVP, MVVM має зв'язуючий пристрій, який автоматизує зв'язок між поданням і його пов'язаними властивостями в моделі перегляду. Модель перегляду була описана як стан даних у моделі. Основною відмінністю між моделлю представлення та презентатором у шаблоні MVP є те, що презентатор має посилання на подання, тоді як модель представлення не має. Замість цього, представлення безпосередньо прив'язується до властивостей

моделі представлення для надсилання та отримання оновлень. Для ефективного функціонування, потрібна технологія прив'язки або генерування коду шаблону для виконання прив'язки.

- Binder (зв’язувач). Декларативні дані і прив'язка команд неявні в

шаблоні MVVM. У стеку рішень Microsoft зв’язувач це мова розмітки під назвою XAML. Зв'язувач звільняє розробника від зобов'язання писати шаблонну логіку для синхронізації моделі перегляду і перегляду.

MVVM було розроблено для використання функцій прив'язки даних у WPF (Windows Presentation Foundation) для кращого полегшення поділу розробки шарів перегляду від решти шаблону, видаляючи практично весь код графічного інтерфейсу ("code-behind") з шару перегляду. Замість того, щоб вимагати від розробників користувальницького досвіду (UX) писати код GUI, вони можуть використовувати мову розмітки фреймворку (наприклад, XAML) і створювати прив'язки даних до моделі перегляду, яка написана і підтримується розробниками програм. Поділ ролей дозволяє інтерактивним дизайнерам зосередитися на потребах UX, а не на програмуванні бізнес-логіки. Таким чином, шари програми можуть бути розроблені в декількох робочих потоках для підвищення продуктивності. Навіть коли один розробник працює на всій основі коду, належне поділ перегляду від моделі є більш продуктивним, оскільки користувальницький інтерфейс зазвичай змінюється часто і в кінці циклу розробки на основі зворотного зв'язку з кінцевим користувачем.

Шаблон MVVM намагається отримати обидві переваги поділу функціонального розвитку, що надаються MVC, при цьому використовуючи переваги прив'язки даних і фреймворка за допомогою зв'язування даних, максимально наближених до чистої прикладної моделі. Для перевірки вхідних даних використовуються функції перевірки даних, моделі перегляду та перевірки даних будь-яких бізнес-шарів. Результатом є те, що модель і структура обробляють якомога більше операцій, усуваючи або зводячи до мінімуму логіку програми, яка безпосередньо керує переглядом (наприклад, за кодом).

Оскільки, робота клієнтського застосунку полягає, по суті, у відображені та

зміні поточного стану всієї системи, для керування цим глобальним станом було використано архітектурний підхід Flux.

Flux - це архітектурна схема, запропонована компанією Facebook для будівництва СПА. Він пропонує розділити додаток на наступні частини:

- Store (сховище).
- Dispatcher (відправник).
- Views (представлення).
- Actions creators / Actions (творці подій / події).

Сховище (Store) - керує станом. Він може зберігати як стан домену, так і стан інтерфейсу користувача. Сховище і стан є різними поняттями. Стан - це значення даних. Сховище це, по суті, об'єкт поведінки, який керує станом за допомогою методів.

Відправник (Dispatcher) - це один об'єкт, який передає дії та події до всіх зареєстрованих сховищ. Сховища повинні реєструватися для подій при запуску програми.

Перегляд - це компонент інтерфейсу користувача. Він відповідає за візуалізацію інтерфейсу користувача та обробку взаємодії з користувачем. Перегляди знаходяться в деревній структурі.

Представлення (Views) прослуховують зміни сховища і змінюються відповідно до них. Погляди представлень не підключаються до диспетчера чи сховищ. Вони спілкуються лише через власні властивості. Види контейнерів підключені до магазинів і диспетчера. Вони слухають події зі сховищ і надають дані для компонентів презентації. Вони отримують нові дані, використовуючи загальнодоступні методики сховищ, а потім передають ці дані по дереву переглядів. Погляди контейнера відправляють дії у відповідь на ітерацію користувача. Коли дія прийде, вона передасть цю дію всім зареєстрованим сховищам.

Дія (Action) - це звичайний об'єкт, який містить всю інформацію, необхідну для виконання цієї дії. Дії мають властивість типу, що ідентифікує тип дії. Як об'єкти дії рухаються навколо програми, я пропоную зробити їх незмінними. Дії можуть надходити з різних місць. Вони можуть виходити з представлення в результаті

взаємодії з користувачем або з інших місць, наприклад, з коду ініціалізації, де дані можуть бути взяті з веб-API, а дії звільняються для оновлення переглядів. Дія може надходити від таймера, який вимагає оновлення екрану.

Загальна схема взяємодії основних складових сутностей, які виділяє паттерн Flux зображена на рисунку 4.5.

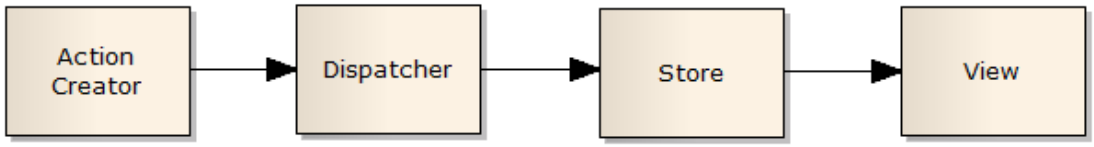

Рисунок 4.5 – Паттерн Flux

4.4 Концептуальна модель бази данних

База даних системи складається з трьох взаємопов’язаних таблиць нереляційної бази даних, які створюють єдиний інформаційний простір для зберігання та отримання доступу до даних.

Головною таблицею є “Користувач”, яка містить у собі інформацію про користувача системи, основну інформацію про нього та індивідуальні налаштування.

Таблиця “Ресурс” містить основну інформацію про кожен конкретний інформаційний ресурс, його назву, дату створення, власника, опис.

Таблиця “Директорія” містить інформацію про конкретну директорію: назву, батьківську директорію, дату створення.

Таблиця “Файл” містить інформацію про файл, завантажений до хмарного сховища Google Drive а саме унікальний ідентифікатор файлу, за допомогою якого можна здійснити доступ до файлу, який уже було завантажено до хмарного сховища при створенні ресурсу.

4.5 Опис таблиць бази даних

Для доступу до даних із бази даних для кожної таблиці створюється набір моделей за допомогою ORM Mongoose, які є об’єктним відображенням таблиці в базі даних.

База даних системи реалізована за допомогою MongoDB. Розглянемо більш детально структури кожної із таблиць бази даних системи для надання доступу до інформаційних ресурсів.

Детальна інформація про їх структури (ім’я, тип і розмір поля, опис поля) приведена у таблицях 4.3 — 4.6.

Таблиця 4.3. Структура таблиці “Користувач”

Ім’я поля	**Тип поля**	**Опис поля**
_id	ObjectId	Первинний ключ
fullname	String	Логін користувача
email	String	Адреса електронної пошти
password	String	Хеш паролю
scopes	String[]	Масив прапорців доступу

Таблиця 4.4. Структура таблиці “Ресурс”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
_id	ObjectId	Первинний ключ
name	String	Назва ресурсу
description	String	Опис ресурсу
file	ObjectId	Посилання на файл
owner	ObjectId	Посилання на власника ресурсу
folder	String	Посилання на директорію

Таблиця 4.5. Структура таблиці “Директорія”

Ім’я поля	**Тип і розмір поля**	**Опис поля**

_id	String	Первинний ключ
name	String	Назва директорії
owner	ObjectId	Посилання на власника директорії
parent	ObjectId	Посилання на батьківську директорію

Таблиця 4.6. Структура таблиці “Файл”

Ім’я поля	**Тип і розмір поля**	**Опис поля**
_id	String	Первинний ключ
filename	String	Назва файлу
gdFileId	String	Унікальній ідентифікатор файлу в хмарному сховищі Google Drive
extension	String	Розширення файлу

4.6 Висновки до розділу

В цьому розділі ми розглянемо опис розробленого програмного продукту та особливості програмної реалізації модулів розробленої системи:

- Серверна частина;
- клієнтська частина;
- база даних.

В цьому розділі було розглянуто опис розробленого програмного продукту та особливості реалізації модулів розробленої системи:

- Серверну частину було побудовано на основі архітектурного принципу REST, що дозволило будувати надійну, масштабовану та зрозумілу архітектуру.
- Для клієнтської частини було вибрано фреймворк Vue.js та надбудову над мовою Javascript - TypeScript. Це дало змогу використовувати при розробці строгу типізацію, що дозволили уникнути більшості помилок уже на стадії розробки програмного продукту.

- Для клієнтської частини було вибрано шаблон проектуваня MVVM, який реалізує вибраний фреймворк - Vue.js. Це дало змогу правильно розставити границі модулів клієнтської частини, що, в свою чергу, дозволили будувати розширюваний та легкий для підтримки застосунок.
- Як базу даних було вибрано нереляційну базу даних, оскільки вона дозволяє зберігати великий об’єм неструкторованих даних, дозволяє використовувати хмарні сховища даних та значно пришвидшує розробку програмного продукту.

5 РОБОТА КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

Розроблена програмний комплекс розроблений з використанням веб-технологій і тому працює в браузерах, які підтримують актуальні веб-стандарти.

5.1 Інсталяція та системні вимоги

Оскільки застосунок працює з використанням веб-технологій, тому від не потребує встановлення на пристрій користувача. Проте, для використання необхідний веб-браузер, який підтримує актуальні веб-стандарти.

5.2 Інструкція з використання програмного продукту

При вході на клієнтський додаток, користувачеві необхідно авторизуватися в системі щоб почати працювати в системі. Для цього він має ввести свою електронну пошту, яка була вказана при реєстрації раніше та пароль. Пара електронна пошта та пароль є строго унікальною для кожного користувача системи та однозначно його ідентифікує на сервері. Для поля Email встановлена валідація, так що користувач може ввести електронну пошту, яка точно відповідає поставленим вимогам. Форма реєстрації зображена на рисунку 5.1.

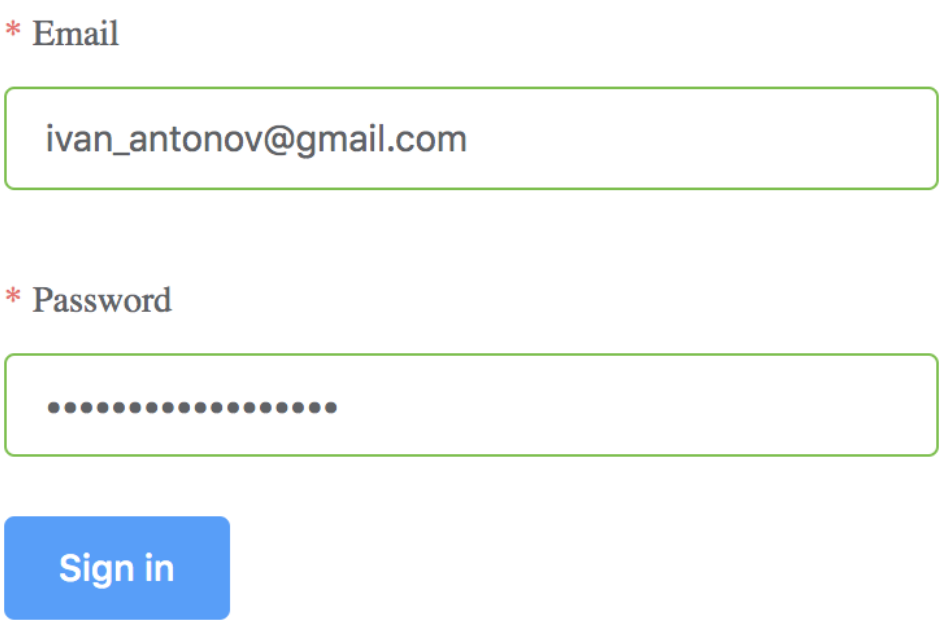

Рисунок 5.1 - Форма авторизації

Якщо користувач користується системою вперше, він має створити власний обліковий запис. Для цього йому потрібно ввести своє повне ім’я, електронну пошту та пароль. Після натиснення кнопки “Sign up” (Зареєструватись) - створюється обліковий запис користувача в системі. Тепер він може повернутись на сторінку логіна та, ввівши електронну пошту та пароль, вказані при реєстрації, отримати доступ до реєстру. Для полів форми реєстрації визначено алгоритм валідації. Для в поле Full Name, можливо ввести лише нечислові значення, Email – може бути лише валідною адресою електронної пошти. Якщо хоча б одне з полів форми, не відповідає вимогам валідації – кнопка Sign Up є неактивною, тобто натиснення на неї не дасть ніякого результату. Форма реєстрації зображена на рисунку 5.2

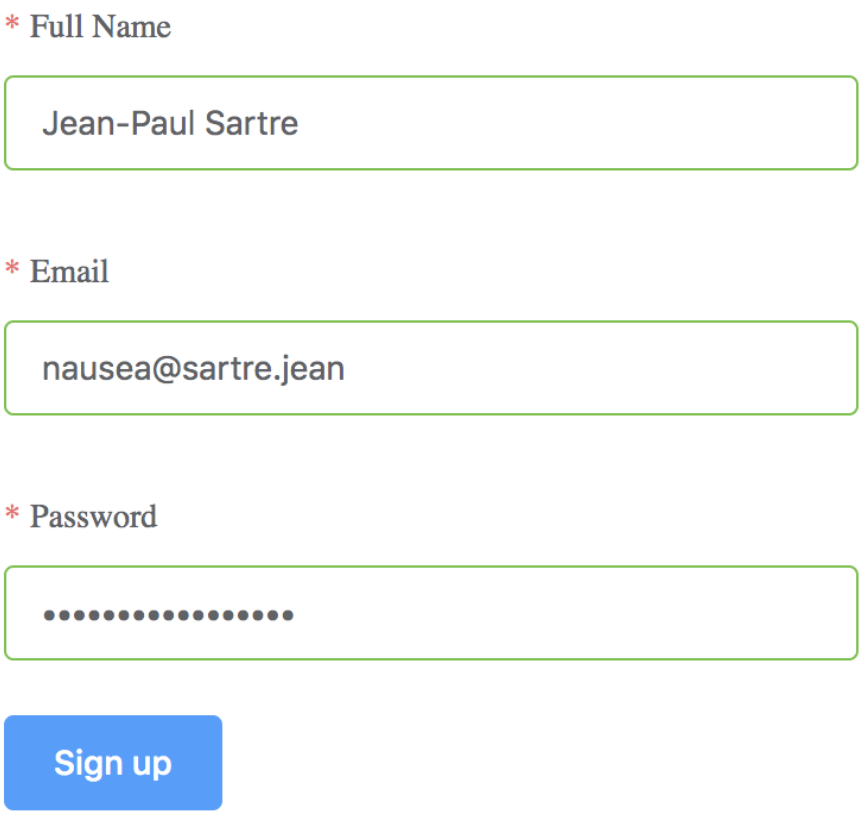

Рисунок 5.2 - Форма реєстрації

Після вдалої аутентифікації, користувач попадає на головну сторінку перегляду ресурсів (рисунок 5.3). В верхній частині екрану розміщений логотип системи, кнопка для виходу “Logout” (Вихід) та вітання.

В лівому сайдбарі представлені зв’язки між директоріями у вигляді дерева. Натиснувши по назві конкретної директорії, користувач попадає на сторінку цієї директорії. Також у верхній частині сайдбару є кнопка, яка дозволяє оновити стан дерева директорій. (Рисунок 5.4)

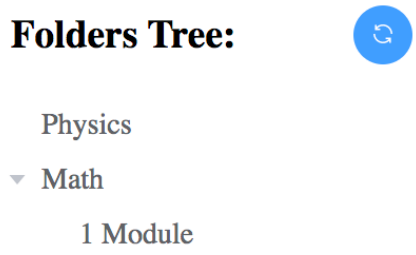

Рисунок 5.4 – Дерево директорій

В центральній частини вікна розміщені, спочатку список директорій (Folders),

а нижче список інформаційних ресурсів (Resources). Кожна з карток ресурсу відображає наступну інформацію:

- Час створення ресурсу;
- назву ресурсу;
- ім’я власника ресурсу;
- назву файлу, на який посилається цей ресурс;
- короткий опис ресурсу, обмежений 128 символами;
- кнопки для виконання дій над ресурсом (видалення, редагування, перехід на сторінку з детальною інформацією про ресурс).

В цій частині також розміщено кнопки, які посилають користувача на сторінки створення інформаційного ресурсу та нової директорії.

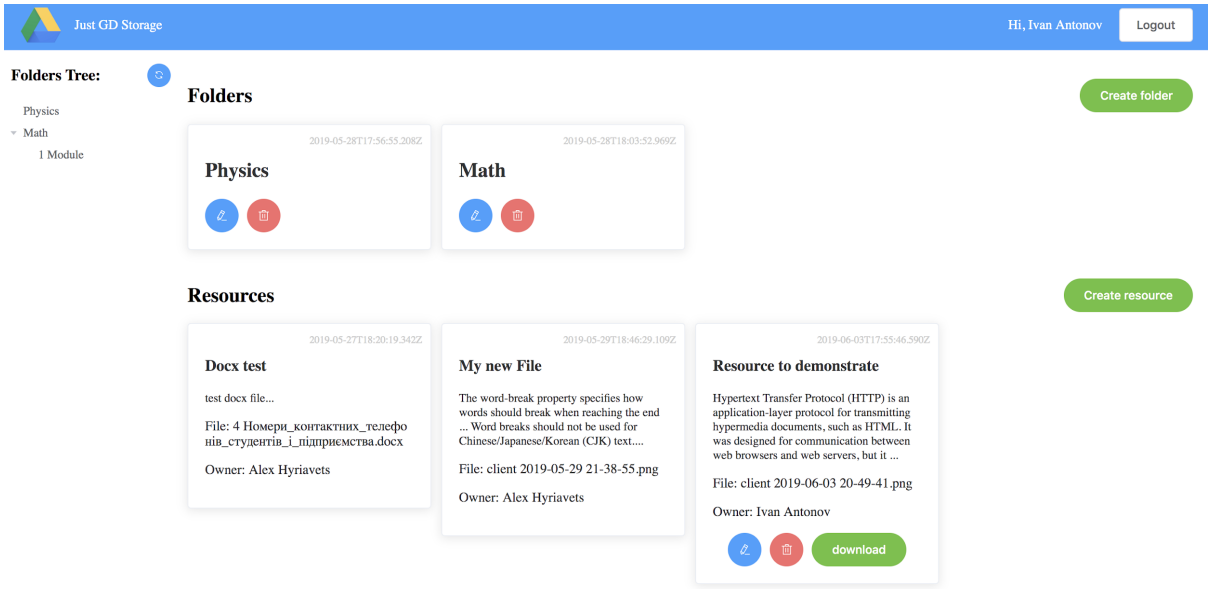

Рисунок 5.3 - Сторінка перегляду інформаційних ресурсів

Для створення нового інформаційного ресурсу, користувач має ввести наступні дані:

- Назву ресурсу;
- опис ресурсу;
- директорію, в якій буде розміщено ресурс;
- завантажити файл з локального комп’ютера.

Та натиснути кнопку “Accept” (Підтвердити). Після чого, завантажений файл буде

збережено на хмарному сервісі Google Drive та його унікальний ідентифікатор буде занесено в базу даних. В разі успіху, користувач побачить відповідне повідомлення. Форма створення інформаційного ресурсу зображена на рисунку 5.4.

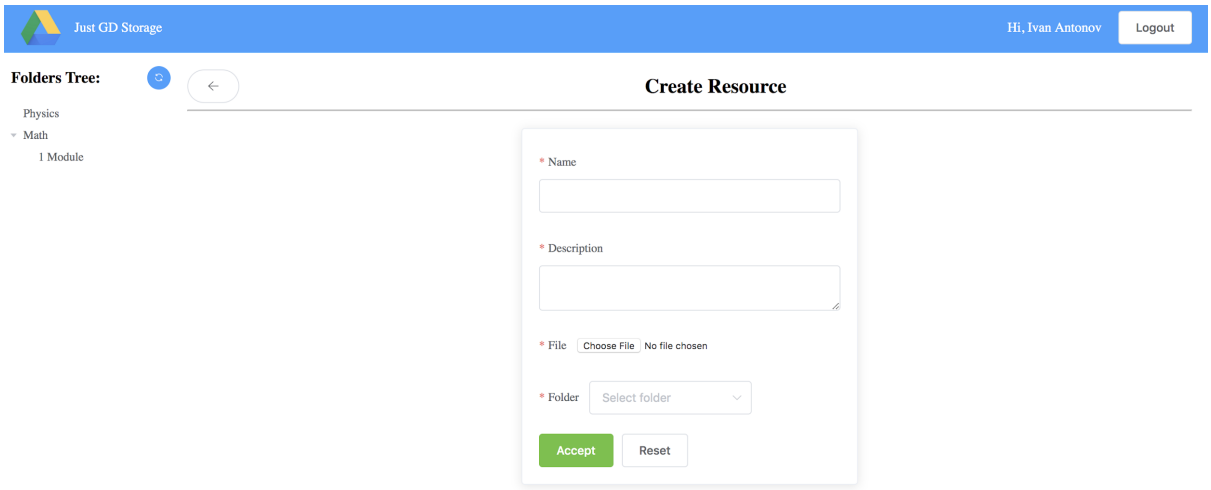

Рисунок 5.4 - Форма створення інформаційного ресурсу

Аналогічно до інформаційного ресурсу, користувач може створювати директорії, з метою структуризації інформації. Для створення нової директорії, необхідно вказати такі дані:

- Назва директорії;
- батьківська директорія;

Та натиснути кнопку “Accept” (Підтвердити). Після чого в базу даних буде занесена інформація про нову директорію і в ній можна буде розміщувати ресурси. Форма створення директорії наведена на рисунку 5.5.

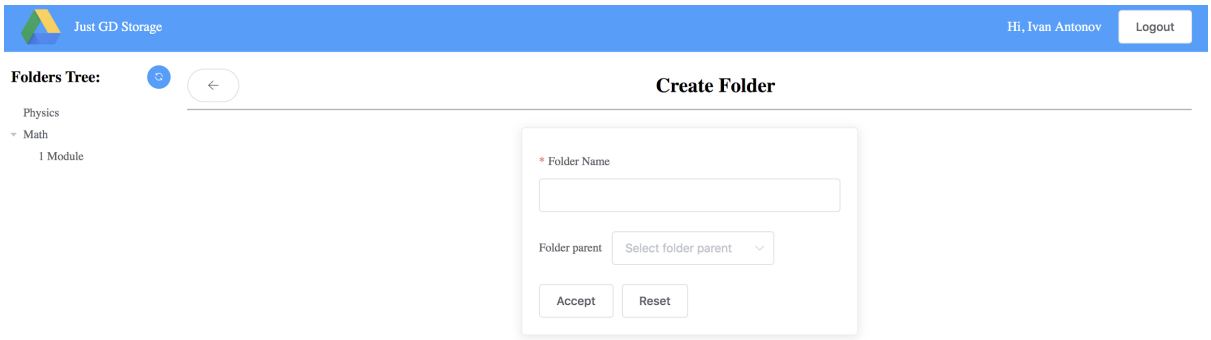

Рисунок 5.5 - Форма створення директорії

При перегляді списку ресурсів, користувачу може стати потрібно переглянути

більш докладнішу інформацію про конкретний інформаційний ресурс. Зробити це можна декількома способами:

1) Натиснувши кнопку “Details” (Деталі) на карточці ресурсу;
2) Клікнути по назві або опису ресурсу на карточці ресурсу.

Після виконання цих дій, користувач попадає на сторінку докладної інформації про інформаційний ресурс. Тут він може побачити наступну інформацію про ресурс:

- Назву;
- розширений опис;
- дату та час створення;
- автора ресурсу;
- назву файлу, на який посилається цей ресурс;

Крім цього, користувач може вивантажити файл на локальний комп’ютер, скористувавшись кнопкою, яка розміщена в правому верхньому куті, “Download” (Завантажити). Після цього, залежно від налаштувань браузера, користувач має вибрати шлях, куди має бути завантажено файл або завантаження файлу почнеться відразу в директорію, вибрану за замовчуванням. (Рисунок 5.6)

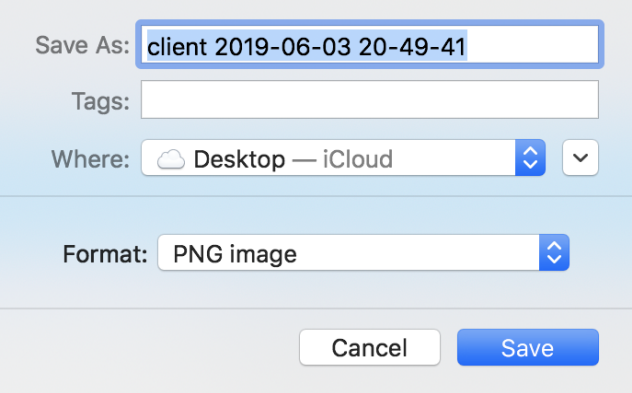

Рисунок 5.6 - Ділогове вікно збереження файлу

Для зручнішої навігації по системі, на сторінці також є кнопка, яка повертає користувача на попередню сторінку, яку він переглядав. Сторінка деталей інформаційного ресурсу зображена на рисунку 5.7.

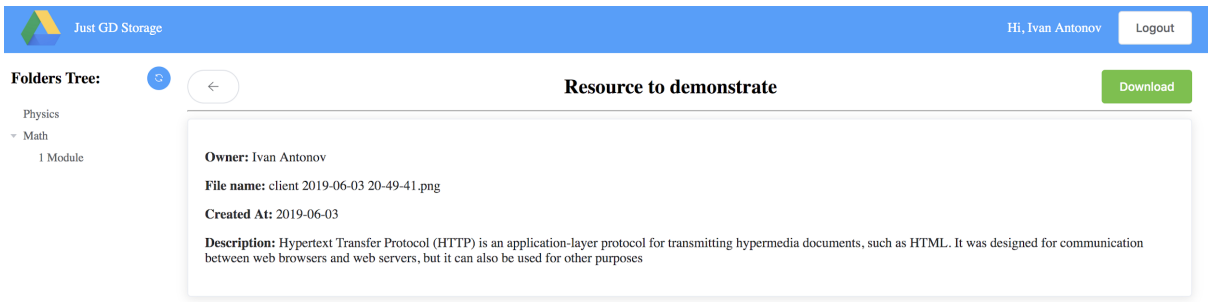

Рисунок 5.7 - Сторінка деталей інформаційного ресурсу

5.3 Висновки до розділу

В цьому розділі було розглянуто алгоритм роботи з системою та показано основні можливості, які система надає кінцевому користувачу. Даний розділ дозволяє користувачеві швидше зрозуміти принципи роботи з системою. Для більшого розуміння були подані знімки основних екранів та покрокові інструкція користування програмою і всіма її модулями.

ВИСНОВКИ

Під час виконання даної дипломної роботи було розроблено веб-сервіс для надання доступу до реєстру інформаційних ресурсів в хмарному сервісі Google Drive.

Сервіс складається з двох основних частин: серверний та клієнтський застосунок.

Для виконання поставленої мети було виконано наступні з поставлених завдань:

1. Проаналізовані поставлені завдання, розроблено план щодо виконання.
2. Було досліджено існуючі систем з надання доступу до реєстру інформаційних ресурсів, виявлено їх переваги та недоліки.
3. Розроблено алгоритм роботи користувача з системою.
4. Розроблено макет дизайну користувацького інтерфейсу.
5. Розроблено загальну архітектуру системи.
6. Створено серверний застосунок, який дозволяє завантажувати файли до хмарного сховища Google Drive, отримувати їх унікальні ідентифікатори та зберігати їх у базі даних системи.
7. Створено клієнтський застосунок, який реалізує користувацький інтерфейс для роботи з реєстром інформаційних ресурсів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ньюкомер Э. Web-сервисы. Для професионалов. — СПб.: Питер, 2003. — 256 с.
2. Lamont I. Google Drive & Docs in 30 Minutes (2nd Edition): The unofficial guide to the new Google Drive, Docs, Sheets & Slides / Ian Lamont., 2015. – 112 с.
3. Документація по Node.Js [Електронний ресурс] – Режим доступу до ресурсу: https://nodejs.org/uk/docs/
4. Elliott E. Programming JavaScript Applications: Robust Web Architecture with Node, HTML5, and Modern JS Libraries / Eric Elliott., 2013.
5. Filipova O. Complete Vue.js 2 Web Development / Olga Filipova., 2016. – 334 с.
6. Duckett J. HTML And CSS: Design And Build Websites / Jon Duckett., 2011. – (John Wiley & Sons).
7. Basarat Ali Syed Beginning Node.js. Apress, 2014. - 308 с.
8. Zakas N. Professional JavaScript for Web Developers / Nicholas Zakas., 2005.
9. Итан Браун Веб-разработка с применением Node и Express. - М. СПб: Издательский дом "Питер", 2016. - 336 с.
10. Нестеренко О.В. Технології інтеграції інформаційних ресурсів інформаційно-аналітичних систем органів державної влади // Науково-технічна інформація. — 2001. — № 4. — С. 3–6.

ДОДАТОК 1

Засоби доступу до реєстру інформаційних ресурсів в хмарному сервісі Google Drive

Специфікація

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТВ51

Аркушів 1

Київ – 2019

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТВ51 19Б 81-1	Гирявець_О. Ю_ТВ51.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТВ-51 19Б 12-1	app.js	Модулі серверної частини
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТВ-51 19Б 12-2	Resources.js	Інтерфейси для роботи з ресурсами
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТВ-51 19Б 12-3	Users.js	Інтерфейси для роботи з користувачами
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТВ-51 19Б 12-4	createResource.js	Контролер для створення ресурсу
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТВ-51 19Б 12-5	uploadFile.js	Контролер для завантаження файлу до Google Drive
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕ ПС_ТВ-51 19Б 13-1	Опис.docx	Опис модуля інтерфейсу клієнтської частини

ДОДАТОК 2

Засоби доступу до реєстру інформаційних ресурсів в хмарному сервісі Google Drive

Текст програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТВ51_19Б 12-1

Аркушів 6

Київ – 2019

```
const express = require('express')
const app = express()
const bodyParser = require('body-parser')
const cors = require('cors')
const mongoose = require('mongoose')

const port = 3001

// Express middlewares
app.use(cors())
app.use(bodyParser.urlencoded({ extended: false }))
app.use(bodyParser.json())
app.use(require('morgan')('dev'))

// MongoDB connection
mongoose.connect('mongodb://localhost/just-gd-storage', {
  useNewUrlParser: true,
})
mongoose.set('debug', true)
const { connection } = mongoose
connection.on('error', console.error.bind(console, 'connection error:'))
connection.once(
  'open',
  console.error.bind(console, 'db connected successfully'),
)

// Routes
app.use(require('./routes'))

app.listen(port, () => console.log(`App listening on port ${port}!`))
```

УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТВ-51 19Б 12-2

```
/* Формування API для роботи з ресурсами */

// Підключення залежностей
const express = require('express')
const router = express.Router()
const multer = require('multer')

// Підготовка локального файлового сховища для
// збереження файлів
const storage = multer.diskStorage({
  destination: './../uploads',
```

```
  filename: function(req, file, cb) {
    cb(null, file.fieldname + '-' + Date.now())
  },
})

const upload = multer({ storage: storage })

// Інтерфейс для взаємодії з клієнтським застосунком
router.get('/', require('./../controllers/resources/getResources'))
router.get(
  '/related',
  require('./../controllers/resources/getRelatedResources'),
)
router.post(
  '/create',
  upload.single('file'),
  require('./../controllers/resources/createResource'),
)
router.put(
  '/edit',
  upload.single('file'),
  require('./../controllers/resources/editResource'),
)
router.post('/remove', require('./../controllers/resources/removeResource'))

// Експорт отриманого модуля
module.exports = router
```



```
/* Формування API для роботи з користувачами */

// Підключення залежностей
const express = require('express')
const router = express.Router()
const auth = require('./../config/auth')

// Інтерфейс для взаємодії з клієнтським застосунком
router.post('/register', require('../controllers/users/createUser'))
router.post('/login', require('../controllers/users/login'))
router.get('/profile', require('../controllers/users/fetchProfile'))

// Експорт отриманого модуля
module.exports = router
```

```
/* Контроллер для створення ресурсу */

// Підключення залежностей
const fs = require('fs')
const path = require('path')
const { Resource } = require('./../../models')
const { getGdApi } = require('../../utils/googleApi')

// Допоміжна фінкція для перевірки рівня доступу
const hasPermissions = permissions => permissions && Boolean(permissions.length)

// Функція контроллера
const createResource = async (req, res) => {
  const { body: newResource } = req

  // Валідація отриманих з запитом даних
  if (
    !newResource.name ||
    !newResource.owner ||
    !hasPermissions(newResource.readPermissions) ||
    !hasPermissions(newResource.editPermissions) ||
    !hasPermissions(newResource.deletePermissions) ||
    !newResource.file
  ) {
    res.status(400).send('Some fields not filled')

    return
  }

  // сторення новго ресурсу
  const resource = new Resource(newResource)

  try {
    // збереження нового ресурсу в базі даних
    await resource.save()

    // повідомляємо користувача про успішне додавання
    // ресурсу в базу даних
    res.send('Resource created successfully')
  } catch (err) {
    // якщо при додаванні ресурсу сталася помилка
    // інформуємо про це користувача
    res.status(400).send('Error during creating resource')
```

```
  }
}

// Експортуємо даний контролер
module.exports = createResource
```



```
/* Завантаження файлу до Google Drive */

// Підключення залежностей
const path = require('path')
const mime = require('mime-types')
const fs = require('fs')
const { File } = require('./../../models')
const { getGdApi } = require('./../../utils/googleApi')

// Функція для отримання розширення файлу
function getFileExtension(filename) {
  if (!filename) return ''

  const splittedFilename = filename.split('.')

  return splittedFilename[splittedFilename.length - 1]
}

// Функція для завантаження файлу
async function uploadFile(req, res) {
  const { file } = req

  // Валідація
  if (!file) {
    res.status(400).send('File not found')
    return
  }

  // Підготовка файлі до завнтаження на гугл драйв
  const fileMetadata = {
    name: file.originalname,
  }
  const media = {
    mimeType: file.mimetype,
    body: fs.createReadStream(file.path),
  }
```

```
// Отримення обєкту аутентифікації
const gdApi = await getGdApi()

// Завантаження на гугл драйв
try {
  const response = await gdApi.files.create(
    {
      resource: fileMetadata,
      media,
      fields: 'id',
    },
    async (err, { data }) => {
      if (err) {
        res
          .status(200)
          .send('Erorr. during uploading file to google drive', err)

        return
      }

      const newDbFile = new File({
        gdFileId: data.id,
        filename: file.originalname,
        extension: getFileExtension(file.originalname),
      })

      await newDbFile.save()

      res.send(newDbFile)
    },
  )
} catch (err) {
  res.status(400).send('Error during uploading a file')
}

// Очищуєм проміжну директорію
const directory = './../uploads'

fs.readdir(directory, (err, files) => {
  if (err) throw err

  for (const file of files) {
    fs.unlink(path.join(directory, file), err => {
      if (err) throw err
```

```
      })
    }
  })
}

module.exports = uploadFile
```

ДОДАТОК 3

Засоби доступу до реєстру інформаційних ресурсів в хмарному сервісі Google Drive

Опис програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТВ51_19Б 13-2

Аркушів 7

Київ – 2019

АНОТАЦІЯ

Додаток надає можливість користувачеві преглядати, створювати, редагувати та налаштовувати рівні доступу до реєстру інформаційних ресурсів в хмарному сервісі Google Drive.

Web-сервіс було розроблено за допомогою платформи Microsoft Visual Studio Code v1.27 з використанням мови програмування JavaScript як для клієнтської, так і для серверної частини. Фрейморком для клієнтськой частини було вибрано Vue.js, а для серверної Express.js.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

Відповідно до теми дипломної роботи, програма має назву «Реєстр інформаційних ресурсів».

Програма працює через браузер та потребує доступу до мережі інтернет, але не потребує встановлення на ПК зайвого ПО, що забезпечує практичність та швидкість.

Система була написана мовою JavaScript з використанням фреймворку Vue.js та Node.js.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Розроблений програмний засіб покликаний вирішити задачу збереження та організації спільного доступу до реєстру інформаційних ресурсів. Система надає користувачам наступні можливості:

- Переглядати інформаційні ресурси;
- структурувати інформаційні ресурси;
- створювати інформаційні ресурси;
- організовувати рівні доступу до інформаційних ресурсів;
- вивантажувати файли на локальний комп’ютер.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Програмний продукт складається з клієнтської та серверної частини.

Серверна частина описує інтерфейс, згідно з яким клієнтська частина, за допомогою AJAX-запитів, здійснює роботу з базою даних та сервісом Google Drive.

Клієнтська частина реалізує користувацький інтерфейс для роботи з реєстром інформаційних ресурсів. Після вдалого проходження аутентифікації, користувач попадає до корневого каталогу системи, в якому відображаються існуючі ресурси та директорії. Користувач може здійснювати навігацію по директоріям та створювати нові директорії. Обмежень на рівні вкладеності директорії система не встановлює.

Після натискання на кнопку «Створити ресурс», користувач попадає на сторінку створення ресурсу, де він має ввести дані про новий ресурс та завантажити файл для цього ресурсу.

Для завантаження ресурсу, користувач має перейти на сторінку деталей ресурсу та натиснути кнопку «Завантажити ресурс». Після чого відкриється діалогове вікно системи, де необхідно буде вибрати шлях, куди буде завантажено файл.

ВИКОРИСТОВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для організації доступу до програмного продукту потрібно мати комп’ютер або ноутбук та встановлений браузер, який підтримує актуально веб-стандарти.

Кінцевим користувачам для роботи з програмою потрібно, щоб на комп’ютері був встановлений браузер, Node.js та Vue.js CLI.